Our Ref. No. 042390.P10795
Express Mail No.: EV323393255US

UNITED STATES PATENT APPLICATION

FOR

INVALIDATING TRANSLATION LOOKASIDE BUFFER ENTRIES IN A VIRTUAL MACHINE (VM) SYSTEM

INVENTORS:

Erik C. Cota-Robles
Andy Glew
Stalinselvaraj Jeyasingh
Alain Kagi
Michael A. Kozuch
Gilbert Neiger
Richard Uhlig

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Blvd., 7th Floor
Los Angeles, CA 90025-1026
(714) 557-3800

BACKGROUND

1. Field of the Invention

[0001] This invention relates to virtual machines. In particular, the invention relates to translation lookaside buffers that support a virtual-machine system.

2. Description of Related Art

[0002] A virtual-machine system is a computer system that includes a virtual machine monitor (VMM) supporting one or more virtual machines (VMs). A Virtual Machine Monitor (VMM) is a software program that controls physical computer hardware and presents programs executing within a Virtual Machine (VM) with the illusion that they are executing on real physical computer hardware. Each VM typically functions as a self-contained platform, controlled by a "guest" operating system (OS), i.e., an OS hosted by the VMM, which executes as if it were running on a real machine instead of within a VM.

[0003] To accomplish this simulation, it is necessary for some operations within a VM (e.g., attempts to configure device hardware) to be trapped and emulated by the VMM, which will perform operations to simulate virtual hardware resources (e.g., a simulated device) to maintain the illusion that the guest OS is manipulating real hardware. Thus, in a virtual-machine system transitions from a VM to the VMM and back will occur with some frequency, depending upon the number of instructions and events that the VMM must emulate.

[0004] In a virtual-memory system, a memory address generated by software (a "virtual" address) is translated by hardware into a physical address which is then used to reference memory. This translation process is called paging, and the hardware used to perform the translation is called the paging hardware. In many virtual-memory systems, the virtual-to-physical address translation is

defined by system software in a set of data structures (called page tables) that reside in memory. Modern virtual-memory systems typically incorporate into a system's central processing unit (CPU) a specialized caching structure, often called a translation lookaside buffer (TLB), which stores information about virtual-to-physical address translations and which can be accessed far more quickly than memory.

[0005] When an OS stops executing one process and begins executing another, it will typically change the address space by directing the hardware to use a new set of paging structures. This can be accomplished using a software or hardware mechanism to invalidate or remove the entire contents of the TLB. More frequent than changes between processes are transitions of control between a process and OS software. Because of this, system performance would suffer significantly if the TLB were invalidated on each such transition. Thus, modern operating systems are typically constructed so that no change of address space is required. One or more ranges of (virtual) memory addresses in every address space are protected so that only the OS can access addresses in those ranges.

[0006] In a virtual-machine system, certain operations within a VM must be trapped and emulated by the VMM. While this is much as an OS supports a user process, the situation here is different. Applications designed to run in user processes are bound by the address-space constraints imposed by the OS. In contrast, software that executes in a VM is not aware that it is being supported by a VMM and thus expects to have access to all memory addresses. For this reason, a VM and its supporting VMM cannot easily share an address space.

[0007] If a VM and its support VMM do not share an address space, then transitions between the VM and the VMM will adversely affect performance because all entries in the TLB must be invalidated on each such transition. Therefore, there is a need to have an efficient technique to allow translations for different address spaces to coexist in the TLB in a VM system.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

[0009] Figure 1 is a diagram illustrating a system in which one embodiment of the invention can be practiced.

[0010] Figure 2 is a diagram illustrating a translation lookaside buffer (TLB) shown in Figure 1 according to one embodiment of the invention.

[0011] Figure 3 is a flowchart illustrating an invalidating policy for the TLB entries shown in Figure 2 according to one embodiment of the invention.

[0012] Figure 4 is a list of operations that cause TLB entry invalidations according to one embodiment of the invention. Such operations are referred to as invalidation operations in the following description.

DESCRIPTION

[0013] One embodiment of the present invention is a technique to invalidate entries in a translation lookaside buffer (TLB). A translation lookaside buffer (TLB) in a processor has a plurality of TLB entries. Each TLB entry is associated with a virtual machine extension (VMX) tag word indicating if the associated TLB entry is invalidated according to a processor mode when an invalidation operation is performed. The processor mode is one of execution in a virtual machine (VM) and execution not in a virtual machine. The invalidation operation belongs to a non-empty set of invalidation operations composed of a union of (1) a possibly empty set of operations that invalidate a variable number of TLB entries, (2) a possibly empty set of operations that invalidate exactly one TLB entry, (3) a possibly empty set of operations that invalidate the plurality of TLB entries, (4) a possibly empty set of operations that enable and disable use of virtual memory, and (5) a possibly empty set of operations that configure physical

address size, page size or other virtual memory system behavior in a manner that changes the manner in which a physical machine interprets the TLB entries.

[0014] In the following description, numerous specific details are set forth. However, it is understood that embodiments of the invention may be practiced without these specific details. In other instances, well-known circuits, structures, and techniques have not been shown in order not to obscure the understanding of this description.

[0015] Existing physical machines that support VM operation control the execution environment of a VM using a structure herein referred to as a Virtual Machine Control Structure (VMCS). The VMCS is stored in a region of memory and contains, for example, state of the guest, state of the VMM, and control information indicating under which conditions the VMM wishes to regain control during guest execution. The one or more processors in the physical machine read information from the VMCS to determine the execution environment of the VM and VMM, and to constrain the behavior of the guest software appropriately.

[0016] In systems using paging, the translation process will require one or more memory references because the paging hardware may need to fetch from memory data from one or more page tables. The virtual-to-physical address translation defined by a set of page tables is called an address space. Typically, operating systems execute each application program, or process, in a separate address space. In some virtual-memory systems, the TLB is largely under the control of the paging hardware. One embodiment of the present invention relates to systems in which the paging hardware determines when to cache in the TLB a virtual-to-physical address translation that has been fetched from the page tables in memory.

[0017] The paging structures that define the address translation reside in memory and may be modified by system software. This may cause translation information cached in the TLB to become out of date. In addition, systems allow software to direct the hardware to use an entirely different set of paging structures

so that the entire contents of the TLB may become out of date. The hardware of some virtual-memory systems may detect such changes and then remove or modify out-of-date translation information while other systems provide software with special instructions to remove such information and still other systems use a combination of the two techniques.

[0018] Figure 1 is a diagram illustrating a computer system 100 in which one embodiment of the invention can be practiced. The computer system 100 includes a processor 110, a host bus 120, a memory control hub (MCH) 130, a system memory 140, an input/output control hub (ICH) 150, a mass storage device 170, and input/output devices 180_1 to 180_K.

[0019] The processor 110 represents a central processing unit of any type of architecture, such as embedded processors, micro-controllers, digital signal processors, superscalar computers, vector processors, single instruction multiple data (SIMD) computers, complex instruction set computers (CISC), reduced instruction set computers (RISC), very long instruction word (VLIW), or hybrid architecture. In one embodiment, the processor 110 is compatible with the Intel Architecture (IA) processor, sometimes referred to as IA-32. The processor 110 typically contains a number of control registers to support memory management tasks such as virtual memory and cache memory. These tasks may include paging and segmentation. The processor 110 also has a cache unit 117. The cache unit 117 has a translation lookaside buffer (TLB) 119. In one embodiment, the processor 110 is compatible with the Intel Architecture (IA) processor, has control registers and a TLB and further supports a Virtual Machine eXtension (VMX) mode. The VMX mode is a mode whereby all virtualization holes in the processor's instruction set are trapped. Software in this new mode executes with as many protection rings (e.g., 4) and with the same paging protection as it would ordinarily have, but whenever privileged software in a VM attempts to change machine state that is virtualized (e.g., mask interrupts) a variety of hardware and software techniques are used by hardware (e.g., the processor), software (e.g., the VMM) or both to provide software in the VM with the illusion that it has effected

the change to the actual hardware state when in fact only the model presented to the VM has changed state.

[0020] The control registers include a first control register CR_PA 112 and a second control register CR_PM 114. The CR_PA register contains the (physical) address of the currently active page table while the CR_PM register contains various "paging_mode" fields such as a "page mode" bit to enable paging. In one embodiment of the invention these control registers are mapped onto the control registers with the same functionality in the Intel Instruction Set Architecture as follows: CR_PA 112 is CR3, and some fields of CR_PM are located in CR0 and others in CR4. Specifically, the CR0 register has two control bits from CR_PM: protected mode enable (PE) and page mode (PG), and the CR4 register has three control bits from CR_PM: Page Size Extension (PSE), Page Global Enable (PGE), and Physical Address Extension (PAE).

[0021] In another embodiment, the CR_PM 114 register has a new control word of 1 or more bits in size that is only configurable by software when the processor is not in VMX mode: Translation Lookaside Buffer Virtual Machine eXtension (TLBVMX). In an alternative embodiment, CR_PM 114 register has two new control words of identical size of 1 or more bits that are only configurable by software when the processor is not in VMX mode: Translation Lookaside Buffer Virtual Machine eXtension (TLBVMX word) and Invalidation Translation Lookaside Buffer Virtual Machine eXtension (TLBVMX invalidation word) . In yet another alternative embodiment, CR_PM 114 register has two new control words of identical size of 1 or more bits that are only configurable by software when the processor is not in VMX mode: Translation Lookaside Buffer Virtual Machine eXtension (TLBVMX word) and Translation Lookaside Buffer Virtual Machine eXtension Mask (TLBVMX mask word). In still another alternative embodiment, CR_PM 114 register has one new control word of size of 1 or more bits that is fully configurable by software when the processor is not in VMX mode: Translation Lookaside Buffer Virtual Machine eXtension (TLBVMX word) and the Virtual Machine Control Structure (VMCS) has a new field of

identical size, Translation Lookaside Buffer Virtual Machine eXtension Mask (TLBVMX mask field). When the processor is in VMX mode access to the TLBVMX word is controlled by the TLBVMX mask field in the currently active VMCS such that the processor can only access those bits of the TLBVMX word whose corresponding bits are cleared in the TLBVMX mask field in the currently active VMCS. In still another embodiment the TLBVMX mask field is inverted and the condition for access is inverted such that access to bits of the TLBVMX word is only allowed for bits that are set in the TLBVMX mask field in the currently active VMCS.

[0022] The host bus 120 provides interface signals to allow the processor 110 to communicate with other processors or devices, e.g., the MCH 130. The host bus 120 may support a uni-processor or multiprocessor configuration. The host bus 120 may be parallel, sequential, pipelined, asynchronous, synchronous, or any combination thereof.

[0023] The MCH 130 provides control and configuration of memory and input/output devices such as the system memory 140 and the ICH 150. The MCH 130 may be integrated into a chipset that integrates multiple functionalities such as the isolated execution mode, host-to-peripheral bus interface, memory control. For clarity, not all the peripheral buses are shown. It is contemplated that the system 100 may also include peripheral buses such as Peripheral Component Interconnect (PCI), accelerated graphics port (AGP), Industry Standard Architecture (ISA) bus, and Universal Serial Bus (USB), etc.

[0024] The system memory 140 stores system code and data. The system memory 140 is typically implemented with dynamic random access memory (DRAM) or static random access memory (SRAM). The system memory may include program code or code segments implementing one embodiment of the invention. The system memory includes a virtual machine (VM) module 145 and a virtual machine monitor (VMM) module 148. The VM and VMM 145 and 148 may also be implemented by hardware, software, firmware, microcode, or any

combination thereof. The system memory 140 may also include other programs or data which are not shown, such as one or more guest operating systems, as well as sets of page tables 190_1 to 190_L. The sets of page tables 190_1 to 190_L may be created and maintained by software and reside in system memory 140 or they may be implemented and maintained by hardware, firmware, microcode, or any combination thereof. There may be one or more sets of page tables for the Virtual Machine Monitor (VMM) and one or more sets of page tables for each of one or more Virtual Machines (VMs).

[0025] The ICH 150 has a number of functionalities that are designed to support I/O functions. The ICH 150 may also be integrated into a chipset together or separate from the MCH 130 to perform I/O functions. The ICH 150 may include a number of interface and I/O functions such as PCI bus interface, processor interface, interrupt controller, direct memory access (DMA) controller, power management logic, timer, universal serial bus (USB) interface, mass storage interface, low pin count (LPC) interface, etc.

[0026] The mass storage device 170 stores archive information such as code, programs, files, data, applications, and operating systems. The mass storage device 170 may include compact disk (CD) ROM 172, floppy diskettes 174, and hard drive 176, and any other magnetic or optic storage devices. The mass storage device 170 provides a mechanism to read machine-readable media.

[0027] The I/O devices 180_1 to 180_K may include any I/O devices to perform I/O functions. Examples of I/O devices 180_1 to 180_K include controller for input devices (e.g., keyboard, mouse, trackball, pointing device), media card (e.g., audio, video, graphics), network card, and any other peripheral controllers.

[0028] Elements of one embodiment of the invention may be implemented by hardware, firmware, software or any combination thereof. The term hardware generally refers to an element having a physical structure such as electronic, electromagnetic, optical, electro-optical, mechanical, electro-mechanical parts, etc. The term software generally refers to a logical structure, a method, a procedure, a

program, a routine, a process, an algorithm, a formula, a function, an expression, etc. The term firmware generally refers to a logical structure, a method, a procedure, a program, a routine, a process, an algorithm, a formula, a function, an expression, etc. that is implemented or embodied in a hardware structure (e.g., flash memory). Examples of firmware may include microcode, writable control store, and micro-programmed structure. When implemented in software or firmware, the elements of an embodiment of the present invention are essentially the code segments to perform the necessary tasks. The software/firmware may include the actual code to carry out the operations described in one embodiment of the invention, or code that emulates or simulates the operations. The program or code segments can be stored in a processor or machine accessible medium or transmitted by a computer data signal embodied in a carrier wave, or a signal modulated by a carrier, over a transmission medium. The "processor readable or accessible medium" or "machine readable or accessible medium" may include any medium that can store, transmit, or transfer information. Examples of the processor readable or machine accessible medium include an electronic circuit, a semiconductor memory device, a read only memory (ROM), a flash memory, an erasable ROM (EROM), a floppy diskette, a compact disk (CD) ROM, an optical disk, a hard disk, a fiber optic medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, Intranet, etc. The machine accessible medium may be embodied in an article of manufacture. The machine accessible medium may include data that, when accessed by a machine, cause the machine to perform the operations described in the following. The machine accessible medium may also include program code embedded therein. The program code may include machine readable code to perform the operations described in the following. The term "data" here refers to any type of information that is encoded for machine-readable purposes. Therefore, it may include program, code, data, file, etc.

[0029] All or part of an embodiment of the invention may be implemented by hardware, software, or firmware, or any combination thereof. The hardware, software, or firmware element may have several modules coupled to one another. A hardware module is coupled to another module by mechanical, electrical, optical, electromagnetic or any physical connections. A software module is coupled to another module by a function, procedure, method, subprogram, or subroutine call, a jump, a link, a parameter, variable, and argument passing, a function return, etc. A software module is coupled to another module to receive variables, parameters, arguments, pointers, etc. and/or to generate or pass results, updated variables, pointers, etc. A firmware module is coupled to another module by any combination of hardware and software coupling methods above. A hardware, software, or firmware module may be coupled to any one of another hardware, software, or firmware module. A module may also be a software driver or interface to interact with the operating system running on the platform. A module may also be a hardware driver to configure, set up, initialize, send and receive data to and from a hardware device. An apparatus may include any combination of hardware, software, and firmware modules.

[0030] One embodiment of the invention may be described as a process which is usually depicted as a flowchart, a flow diagram, a structure diagram, or a block diagram. Although a flowchart may describe the operations as a sequential process, many of the operations can be performed in parallel or concurrently. In addition, the order of the operations may be re-arranged. A process is terminated when its operations are completed. A process may correspond to a method, a program, a procedure, a method of manufacturing or fabrication, etc.

[0031] Figure 2 is a diagram illustrating a translation lookaside buffer (TLB) 119 shown in Figure 1 according to one embodiment of the invention. The TLB 119 includes a storage 210 and an invalidating mechanism 240.

[0032] The storage 210 is a fast memory organized with two fields TLB entry 220 and virtual machine extension (VMX) tag word 230. For some integer

N the TLB 220 has N TLB entries 220_1 to 220_N and the VMX tag word array 230 has N VMX tag words 230_1 to 230_N. Each of the N TLB entries 220_1 to 220_N has an associated VMX tag word. Each of the N TLB entries 220_1 to 220_N is referred to as a TLB translation or translation for short. Each of the VMX tag words 230_1 to 230_N indicates if the associated TLB entry is invalidated according to a processor mode when an invalidation operation is performed. When a VMX tag word is non-zero, the corresponding TLB entry is referred to as a VMX translation. When a VMX tag word is zero (0), the corresponding TLB entry is referred to as a non-VMX translation.

[0033] At any given time, the processor may be in VMX mode. VMX mode is the mode when the processor is executing guest code in a VM. The processor is not in VMX mode when the VMM is executing. In one embodiment, the VMM operates in the protected mode of operation of an Intel Architecture processor. The VMM may create several VMs and each of the VMs operates in VMX mode. The processor mode may be set by a hardware circuit or by a configuration word programmed by software. Each of the VMX tag words 230_1 to 230_N may correspond to a VM created by the VMM. In one embodiment, the VMX tag word field 230 has a word size of one bit.

[0034] As noted in the preceding paragraph, in one embodiment, the VMX tag word is a single bit, so it is always 1 when in VMX mode and 0 when not in VMX mode (or vice versa). Note that the active level of logical 1 or 0 is for illustrative purposes. Other active levels may be used. A bit is said to be asserted when it is set to its active level and to be negated when it is set to the complement of its active level. In the Intel Architecture, the value of this bit could be stored in a control register, a model-specific register, a new architectural register, or a new register. In fact, the value of this bit need not be visible at all and the Translation Lookaside Buffer Virtual Machine eXtension (TLBVMX word) need not exist *per se* since the necessary information can be derived automatically from other processor state (i.e., 1 when in VMX mode, 0 when not, or vice versa). In an alternative embodiment, the VMX tag word is larger than one bit and it is then set

to the new value of the TLBVMX control word, which corresponds to the particular VM being executed, as the processor transitions into VMX mode. To maintain proper operation of the processor, the VMM would naturally prevent direct accesses by guest OSes to the TLBVMX control word

[0035] In another embodiment, the VMX tag word field 230 has a word size of greater than one bit, which we denote by M, and distinct VMX tag word values, excepting the zero (0) value that is reserved for the VMM, are assigned to VMs as they are created by the VMM until the largest possible value is reached, said value being shared by all subsequently created VMs (e.g., if M is 8 then 0 is for the VMM, 1 through 6 are for the first 6 VMs created and 7 is shared by all additional VMs). In another embodiment, the VMX word field 230 again has a word size of greater than one bit and multiple distinct VMX tag word values are reserved for one or more VMMs and the remaining values are assigned to VMs as they are created by the VMMs until the largest possible value is reached, said value being shared by all subsequently created VMs.

[0036] Some embodiments of this invention may support a TLBVMX word 116 in a new or existing control register. The width of this field is the same as the width of the VMX tag words 230 associated with the TLB entries. It determines the number of different address spaces whose translations may be cached in the TLB at any one time. Some embodiments may report the width of this field to software through some mechanism. For example, IA-32 processors report capabilities such as this through the CPUID instruction or through capability registers. Any such mechanism might be used to report the width of the TLBVMX word supported by a CPU.

[0037] Some virtual-machine systems may support layers of virtual machine monitors (VMMs). For example, a single VMM ultimately controls the CPU. This "root" VMM may support, in guest VMs, other VMMs ("guest" VMMs) that may themselves support guest VMs. The support for layering may be provided by software, hardware, or a combination of the two. For systems based

on embodiments that support the VMX tag words defined in this invention, the multiple VMMs (e.g., root and guest VMMs) may all seek to manage the address spaces of their respective guests using the CPU's TLBVMX word. For embodiments that support a single TLBVMX word, this represents a challenge.

[0038] In one embodiment of the invention, the root VMM partitions the bits in the TLBVMX word into those that it controls and those whose control is yielded to its guest VMMs. If the width (in bits) of the CPU's TLBVMX word is M and the root VMM wants to support 2^L-1 guest VMMs, where $L < M$, then the root VMM can assign to each of its guest VMMs a unique value in the range 1 to 2^L-1. Whenever the guest VMM with value I ($1 \leq I \leq 2^L-1$) is running, the root VMM will ensure that the high L bits of the CPU's TLB tag word will contain the value I. These bits will be given the value 0 only for (1) the root VMM or (2) guests of the root VMM that are not themselves VMMs. The root VMM will present to each guest VMM (through a capability reporting mechanism described above) the abstraction of a VM with a virtual CPU in which the width of the TLBVMX word is M-L. The guest VMMs will each be allowed to control only the low M-L bits of the TLBVMX word. Each such guest can thus support up to $2^{M-L}-1$ guests whose address spaces can concurrently use the TLB.

[0039] This embodiment requires that the root VMM is able to (a) control the reporting of the width of the TLBVMX word to its guest VMMs, and (b) prevent its guest VMMs from modifying or reading selected bits of the TLBVMX word.

[0040] Both of these capabilities are already supported in some virtual-machine systems. For example, in IA-32 systems, it is sufficient to ensure that guest attempts to execute the CPUID instruction or to read capability registers or to read or write control registers cause transitions to the root VM monitor (VMM). The root VMM can then emulate the relevant guest instruction, presenting to its guest the values it desires (e.g., in this case, it would report to its guest that the width of the TLBVMX word is M-L instead of M).

[0041] Trapping to the root VMM from a higher level guest VMM so that the root VMM can emulate a single guest instruction is expensive but, since that reporting the width of the TLBVMX word should be an infrequent operation, this manner of support should not adversely affect performance. On the other hand, guest VMMs will be modifying and/or reading bits in the TLBVMX word frequently as they do VM entrances to their guest VMs.

[0042] In an alternative embodiment of the invention, the processor provides a TLBVMX mask word to provide bit-by-bit capability to mask and shadow fields for control registers to constrain the ability of guest software to read and modify control registers Thus software in a VM is only allowed to access those bits of the TLBVMX word whose corresponding bits are cleared in the TLBVMX mask word. For example, if the TLBVMX word is 10100000 and the TLBVMX mask word is 11110000 then a read by software of the TLBVMX word might return 0000 and software can set the TLBVMX word to any desired value from 10100000 to 10101111 by simply writing the low order bits, the high order bits are automatically supplied by the logical AND of the TLBVMX mask word and the TLBVMX word.

[0043] To efficiently support a layered virtualization architecture, the processor provides a hardware-managed stack of TLBVMX mask words that work as follows. Incident to a VM entrance a VMM can specify a new value of the TLBVMX mask word for the guest (whether VMM or not) but the new value will be automatically logically ORed by the processor with the old value of the TLBVMX mask word so that the logical AND of the new value and the old value of the TLBVMX mask word will be equal to the old value. The old value is automatically restored from the hardware-managed stack upon a VM exit. In the new guest the TLBVMX mask word constrains the space of possible TLBVMX word values. For example, continuing the above example, if the (intermediate) VMM sets the TLBVMX word to 10101100 and specifies that the new TLBVMX mask word will be 1100 then upon VM entrance the TLBVMX mask word is

actually set to 11111100 and the TLBVMX word for the guest will be limited to the values 10101100 through 10101111.

[0044] In one embodiment of the invention, software specifies the new TLBVMX mask word value for the VM by loading an appropriate field in a Virtual Machine Control Structure In yet another embodiment of the invention, the TLBVMX mask word is replaced by a TLBVMX inverted mask word which functions identically except that it is the logical NOT of the TLBVMX mask word and may thus be logically NOTed in the above formulas. Storing and using such an inverted mask word could simplify an actual implementation depending on the underlying logic circuitry in the processor.

[0045] Using these techniques, a guest VMM could reserve K bits in the TLBVMX word (of the M-L bits to which it has access) for its own use, presenting to its guest VMM the abstraction of virtual CPU with a TLBVMX word with only M– (L+K) bits. The technique can be used for at most M layers, after which any guest VMMs would be presented with the abstraction of a virtual CPU that does not support a TLBVMX word.

[0046] The invalidation mechanism 240 invalidates the TLB entries according to an invalidation policy 250. The invalidation mechanism 240 may be implemented as a circuit having a control logic consistent with the invalidation policy 250. The control logic provides an efficient way to update or invalidate the TLB entries in a virtual-machine system.

[0047] Figure 3 is a flowchart illustrating a process 300 to implement the invalidation policy 250 for the TLB entries shown in Figure 2 according to one embodiment of the invention.

[0048] Upon START the process 300 determines if the processor is in VMX mode (Block 310). As noted above, in one embodiment of the invention, the VMX tag word is a single bit and the TLBVMX word may exist only logically, its contents being derived automatically from other processor state. In other

embodiments, the TLBVMX word may be more than one bit and exists as a physical field in the CR_PM 114 control register. For all embodiments of the invention, if the processor is in VMX mode, then all new translations in the TLB set the VMX tag word to match the possibly logical value of the TLBVMX word (Block 325) and if the processor is not in VMX mode, then all new translations set the VMX tag word to match the possibly logical value of the TLBVMX word (Block 340).

[0049] Next, if the processor is in VMX mode, the process 330 determines if an operation causing invalidation of one or more TLB entries is performed (i.e., an invalidation operation) (Block 330). For an Intel Architecture processor, examples of such invalidation operations are INVLPG (invalidates only one entry) and loads to the CR3 register (invalidates all entries). If so, the process 300 invalidates the one or more TLB entries, as appropriate, provided that every TLB entry invalidated must have a VMX tag word value that matches the value in the TLBVMX word (which may exist only logically), otherwise the process 300 is terminated.

[0050] Alternatively, if the processor is not in VMX mode, the process 300 determines if an invalidation operation has been performed (Block 345). If so, the process 300 proceeds to invalidate one or more TLB entries, as appropriate to the invalidation operation being performed (e.g., INVLPG on an Intel Architecture processor), providing that the VMX tag word for the entry or entries match according to one of three rules or implementations. The selection of which rule is to be used depends on performance enhancement and implementation risk. In the conservative rule, the process 300 invalidates one or more TLB entries regardless of the VMX tag words (Block 350). In one embodiment of the configurable rule, the process 300 invalidates only those zero or more TLB entries that have the associated VMX tag word matched with the high order M-M' bits of the TLBVMX word (Figure 1) by using the TLBVMX mask word to perform a logical AND on the VMX tag word (Block 372). In another embodiment of the configurable rule, only those zero or more TLB entries having a VMX tag word

set to match the current (possibly logical) value of the TLBVMX invalidation word are invalidated (Block 374). For all rules, if an invalidation operation is not performed, the process 300 is terminated.

[0051] Note that, regardless of any rules above defining behavior both inside and outside VMX mode, a change to the CR_PA 112 register incident to a transition into VMX mode (i.e., VM entrance) or out of VMX mode (i.e., VM exit) does not invalidate any entries in the TLB. This special case is therefore not an invalidation operation.

[0052] Figure 4 is a list of invalidation operations according to one embodiment of the invention. Specifically, for an Intel Architecture processor there are basically five invalidation operations.

[0053] Invalidation operation 1 (Block410): Loading the first register CR3, subject to the setting of the global (G) bit in the TLB entry and other existing TLB behavior such as the width of the TLBVMX word.

[0054] Invalidation operation 2 (Block420): Execution of the page invalidation instruction (e.g., INVLPG instruction in the Intel Architecture)

[0055] Invalidation operation 3 (Block430): Task switching that modifies the first control register CR3.

[0056] Invalidation operation 4 (Block440): Loading the second register CR0 that changes either the PE bit or the PG bit.

[0057] Invalidation operation 5 (Block450): Loading the third register CR4 that changes one or more of the PSE bit, PGE bit, or PAE bit.

[0058] By providing the VMX tag word associated with each TLB entry in the TLB, a VM system can efficiently trap to the VMM and resume execution of the VM without invalidating the TLB. This significantly increases the efficiency of the VMM system.

[0059] While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.